Performance Evaluation of Components Using a Granularity-based Interface Between Real-Time Calculus and Timed Automata

Karine Altisen Yanhong Liu Matthieu Moy
Verimag, Grenoble, France.
{karine.altisen, yanhong.liu, matthieu.moy}@imag.fr

To analyze complex and heterogeneous real-time embedded systems, recent works have proposed interface techniques between *real-time calculus* (RTC) and *timed automata* (TA), in order to take advantage of the strengths of each technique for analyzing various components. But the time to analyze a state-based component modeled by TA may be prohibitively high, due to the *state space explosion* problem. In this paper, we propose a framework of granularity-based interfacing to speed up the analysis of a TA modeled component. First, we abstract fine models to work with event streams at coarse granularity. We perform analysis of the component at multiple coarse granularities and then based on RTC theory, we derive lower and upper bounds on arrival patterns of the fine output streams using the causality closure algorithm of [2]. Our framework can help to achieve tradeoffs between precision and analysis time.

1 Introduction

Modern real-time embedded systems are increasingly complex and heterogeneous. They may be composed of various subsystems, and it is a general practice that some of the subsystems may be power-managed [5], in order to reduce energy consumption and to extend the system life time. Such a subsystem may have multiple running modes. A mode with lower power consumption also implies lower performance levels. Due to real-time requirements, it is thus critical to analyze the system timing performance, but the complexity of this analysis is challenging, especially when it is scaled to large and heterogeneous systems.

Compositional analysis techniques have been presented as a way of tackling the complexity of accurate performance evaluation of large real-time embedded systems. Examples include SymTA/S (Symbolic Timing Analysis for Systems) [9] and modular performance analysis with real-time calculus (RTC) [7]. Various models have also been proposed to specify and analyze heterogeneous components [14]. Each analysis techniques have their own particular strengths and weaknesses. For example, SymTA/S or RTC based analysis can provide hard lower/upper bounds for the best-/worst-case performance of a system, and has the advantage of short analysis time. But typically, they are not able to model complex interactions and state-dependent behavior and can only give very pessimistic results for such systems. On the other hand, state-based techniques, e.g. timed automata (TA) [3], construct a model that is more accurate, and can determine exact best-/worst-case results. But they face the *state explosion* problem, leading to prohibitively high analysis time and memory usage even for a system with reasonable size.

Efforts have been paid to couple different approaches [15, 10, 17, 1], e.g. to combine the functional RTC-based analysis with state-based models. The most general ones are based on interfacing RTC with another existing formalism: in [16], an interfacing technique is proposed to compose RTC-based techniques with state-based analysis methods using *event count automata* [8]. A tool called CATS [6]

QAPL 2010
EPTCS , 2010, pp. 1–18, doi:10.4204/EPTCS..

is just at the beginning of its development. It allows modeling a component using TA within a system described by RTC. A variant of the approach is described in [11], which restricts to convex and concave curves, but potentially uses less clock and may therefore scale better.

In this paper, we follow the line of recent development in interfacing between RTC and state-based models done in CATS, and focus on speeding up the analysis for a power-managed component (PMC) modeled by TA. We adapt the framework to analyze event streams at different granularities. This implies the ability to design the component to be analyzed at coarser granularities: the translation of the component from one granularity to another has to be made automatic. To do so, we focus on a class of timed automata of interest that we call M-TA, on which the translation is possible. M-TA model power-managed components, characterized with different *modes* of computation. We then define a schema of translation at coarser granularity. The schema is formally proven and the whole approach is validated with experimentation: we show that our abstracted model scales far better than the fine-granularity one, with a reasonable loss of precision.
Organization of the paper: in the next section 2, we detail the implementation of the existing interfacing techniques between RTC and TA. In section 3, we give an overview of the framework for granularity-based interfacing and section 4 formally details the granularity change. In section 5, we discuss our targeted PMC, and its fine and coarse TA models; and in section 6 we present some experimental validations. Finally, we make a conclusion and present future work in section 7.

2 Interfacing Timed Automata and Real-Time Calculus

Real-Time Calculus (RTC). The *Real-Time Calculus (RTC)* [18] is a framework to model and analyze heterogeneous system in a compositional manner. It relies on the modeling of timing properties of event streams and available resources with curves called *arrival curves* and *service curves*. A component can be described with curves for its input stream and available resources and some other curves for the outputs. For already-modeled components, RTC gives exact bounds on the output stream of a component as a function of its input stream. This result can then be used as input for the next component.

An *arrival curve* is an abstraction to represent the set of event streams that can be input to (resp. output from) a component; it is expressed as a pair of curves $\xi = (\xi^L, \xi^U)$. For $k \geq 0$, $\xi^L(k)$ and $\xi^U(k)$ respectively provide for any potential stream the lower and upper bounds on the length of the time interval during which *any* k consecutive events can arrive. Let t_i denote the arrival time of the i-th event; t_i may be real (we use continuous time) but the number of events that occurs at t_i is discrete (it is represented by a natural number); we have $\xi^L(k) \leq t_{i+k} - t_i \leq \xi^U(k)$ for all $i \geq 0$ and $k \geq 0$.

Similarly, the processing capacity of a component is specified by a *service curve* $\psi = (\psi^L, \psi^U)$. The length of the time to process any k consecutive events for any potential stream is at least $\psi^L(k)$ and at most $\psi^U(k)$.

Notice that, in the RTC theory, an arrival curve $\alpha = (\alpha^L, \alpha^U)$ (resp. service curve β) is usually expressed in terms of *numbers of events* per time interval. In this paper, we express the arrival curves (ξ) and service curves (ψ) in terms of *length of time interval*, in order to explain better our work. Actually, ξ is a *pseudo-inverse* of α, satisfying $\xi^U(k) = \min_{\Delta \geq 0}\{\Delta | \alpha^L(\Delta) \geq k\}$ and $\xi^L(k) = \max_{\Delta \geq 0}\{\Delta | \alpha^U(\Delta) \leq k\}$ (same for β and ψ).

Timed Automata (TA). A timed automaton [3] is a finite-state machine extended with *clocks*. A clock measures the time elapsed since its last reset and all clocks increase at the same rate. Figure 3(b) shows an example. States are labelled by *invariants* and transitions by *guards and clock resets* (e.g. $\mathbf{t} \leftarrow 0$).

Invariants and guards are conjunctions of lower and upper bounds on clocks and differences between clocks. The automaton can only let time elapse in a state whenever the invariant evaluates to true. A transition may be triggered when the guard evaluates to true and some clocks are reset. Timed automata may be synchronized via binary rendez-vous: for example, the self-loop transition around `Counting` sends the signal **produce?** This means that the transition cannot be fired unless the automaton receives at the same instant the signal **produce!** from another one. Furthermore, we use UPPAAL TA syntax and extensions [4], such as integer counters (e.g. *cost*).

Interfacing TA and RTC

In the following, we show the techniques from CATS [6] for interfacing TA and RTC. The motivation comes from the fact that some components may not be accurately modeled with RTC tools. The idea is then to use TA for the component model, but to be nevertheless able to consider arrival and service curves to characterize inputs and outputs of the system. This implies the use of adapters, as shown in Figure 1, connecting RTC curves (which describe a set of event streams) and the TA (which inputs or outputs a single event stream). From RTC curves to TA, we use a *generator* that inputs events into the component such that the event stream satisfies the input curve $\hat{\xi}$. From TA to RTC, the component outputs events that feed an *observer* which measures the smallest and largest time interval between events, to compute the output curve $\check{\xi}$. Both generators and observers are timed automata; they are composed with the timed automaton *component* and the tool UPPAAL CORA [4] is used to verify the whole. Notice that this framework is convenient whatever be the computational model used for the component (see e.g. ac2lus [1]).

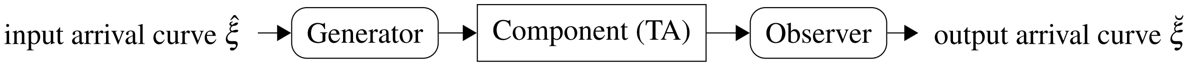

input arrival curve $\hat{\xi}$
Generator
Component (TA)
Observer
output arrival curve $\check{\xi}$

Figure 1: Interfacing RTC and TA

In all this work, RTC curves are assumed to be given by a finite set of N points, namely, (ρ^L, ρ^U) are defined by the points $(\rho^L(i), \rho^U(i))$, for $i \in [1:N]$.

Generator. The goal for the generator is, given an input arrival (resp. service) curve, to be able to generate *any* event stream that satisfies this curve, and only these ones. Figure 2 shows the TA model $Generator(\rho^L, \rho^U, \mathbf{signal})$ which non deterministically generates a sequence of signals called **signal!** that satisfies (ρ^L, ρ^U).

The main idea is to reset a clock whenever an event is emitted. As we need to check any N successive events, we need to memorize only N clocks, that we declare in a circular clock array $\mathbf{y}$ of size N. $\mathbf{y}[k\%N]$ represents the time elapsed since the event k has occurred. We note λ the index of the next event to be generated: then we have to check that the N events before λ comply with the curve. The index of those events is given by the function $getIdx(i) \rightarrow (\lambda - i + N)\%N \quad (1 \leq i \leq N)$. Note that at the beginning, less than N events have been emitted: we introduce a bounded counter θ, from 0 to N that represents the actual number of events to be considered. The constraints that satisfy the curves are then expressed as:

$$\begin{array}{rl} (Check\_Upper) & \mathbf{y}[getIdx(i)] \leq \rho^U(i), \forall i \in [1:\theta] \\ (Check\_Lower) & \mathbf{y}[getIdx(i)] \geq \rho^L(i), \forall i \in [1:\theta] \end{array}$$

$(Check\_Upper)$ expresses an invariant on how much time can elapse (i.e. when an event *must* be generated), whereas $(Check\_Lower)$ qualifies the date from which a new event *may* be emitted, checked before emitting a new signal.

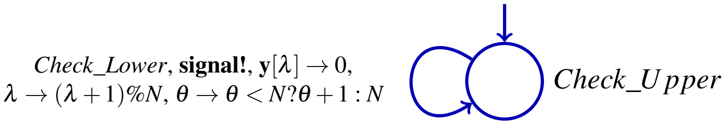

Check_Lower, **signal!**, **y**[λ] → 0,
$\lambda \rightarrow (\lambda+1)\%N$, $\theta \rightarrow \theta < N?\theta+1 : N$
Check_Upper

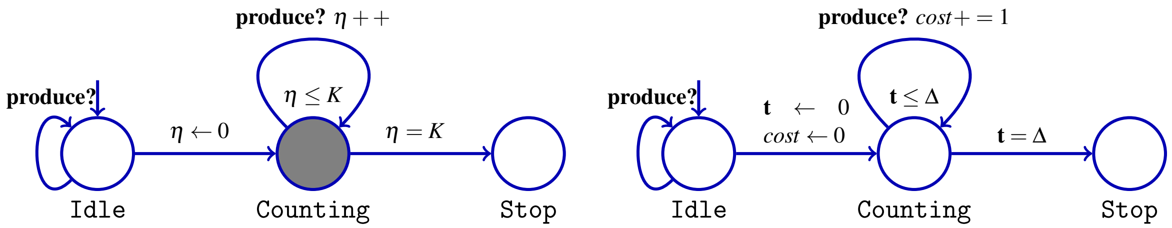
Figure 2: $Generator(\rho^L, \rho^U, \mathbf{signal})$: model of a generator from RTC to TA.

Observer. To compute the output curve (ρ^L, ρ^U), we use a model checking tool with cost optimal reachability analysis [4]. An observer is used to capture the arrival patterns of an event stream: Figure 3(a) computes lower and upper bound for a stream of **produce!** events and for a window of K events, namely $(\rho^L(K), \rho^U(K))$.

The observer non-deterministically transits from the initial state `Idle` to `Counting`: this decides the beginning of the window to be observed. The counter η records the number of **produce!** since the entry in `Counting`. When reaching the state `Stop`, K **produce!** events have been emitted.

The cost for an execution corresponds to the time spent in the `Counting` state (no cost on transition, cost rate of 1 for `Counting` – marked as a grey state in the figure, cost rate of 0 for the other states). Therefore, the cost when reaching the state `Stop` is equal to the time for emitting K events. With a verification engine able to compute the minimum and maximum cost for reaching `Stop`, this provides $\rho^L(K)$,since the K consecutive events were chosen non-deterministically. The computation has to be launched N times (for $K = 1$ to N) to obtain all the points of the curves.

Since the tool we use can not compute a maximum, we use a variant model of the observer shown in Figure 3 (b) for computing the maximum and obtaining $\rho^U(K)$. The principle of the timed automata is the same as the former but it measures the number of events (counter *cost*) that can be emitted during an interval of length Δ (clock **t**). When minimizing the cost, this provides $\alpha^U(\Delta)$. Then, ρ^U is computed as the pseudo-inverse of α^U.



(a) $Observer(K, \mathbf{produce})$ (b)

Figure 3: (a) Model of the observer and (b) its varied model.

How to obtain output arrival curves? We show here how to instantiate the timed automata described above to compute an output arrival curve $\breve{\xi}$ given the input arrival curve $\hat{\xi}$, the input service curve ψ and a component modeled as a timed automaton. The input event stream is labeled by signals **req** and is given by $Generator(\hat{\xi}^L, \hat{\xi}^U, \mathbf{req})$. The input service stream is labelled by signals **serv** and is generated by $Generator(\psi^L, \psi^U, \mathbf{serv})$. We assume that the component inputs **req** and **serv** signals and emits **produce** signals. The output event stream is then analyzed by $Observer(K, \mathbf{produce})$. Those four automata are synchronized and a verification engine is used, to obtain all the points $(\breve{\xi}^L(K), \breve{\xi}^U(K))$, for $K = 1$ to N.

Note that the whole model involves two clock arrays of size N (length of the curve); it also involves two N-counters per generator plus one N-counter for the observer. The size of the model, hence, heavily depends on the number of events to be considered.

3 The Granularity-based Interfacing Framework

Motivations. When model-checking non-trivial systems of TA, one quickly faces the well-known state-explosion problem. Generally speaking, this mainly comes from two sources: clocks and counters. Counters are handled as discrete states in the proof engine and clocks involve the dimensions of the polyhedra (zones) to be computed. As stated above, the CATS approach proposes to model-check (with cost optimization) a model where the numbers of clocks and the order of magnitude of counters are heavily linked with the number of events to be considered and so is the cost to compute the results. Applied to non-trivial components the approach may thus fail to provide any result.

By grouping the events, and refraining from looking at them individually, we can reduce the amount of events analysis and thus both the size of counters and the number of clocks: we can hence get dramatic improvements on the performance. We talk about *fine events* to designate the real events of the system and we group them into *coarse events* which intuitively represent a packet of g fine events, where g is called the granularity of the abstraction. Modeling the system to work with coarse events instead of fine events divides the length of the arrival curve $\hat{\xi}$ and the number of events N to store in buffers by g. By changing the value of g, one can trade performance for accuracy.

Framework. We propose a formal framework of granularity-based interfacing between RTC and TA performance models. The generators for service and arrival curves, the component model and the observer for the output arrival curve deal all with coarse events, which speeds up the analysis.

As we change the granularity, all TA models involved in the framework have to be adapted to deal with the coarse events. For the generators and the observers, this is quite straightforward, but abstracting an arbitrary TA component to a coarse-granularity one would be hard, if at all possible. We focus on a particular class of timed automata that we call M-TA (for "Mode-based Timed-Automata") for which we propose an automatic translation scheme from fine event M-TA to coarse event M-TA. For simplicity of the notations, we define the class of M-TA with an abstract syntax, the semantics of an M-TA being given by the corresponding TA. This is illustrated by (a) and (b) in Figure 4. The correctness of the approach relies on the fact that the coarse-grain translation of an M-TA is an accurate abstraction, in the sense that the lower and upper coarse output curves, analyzed from the coarse model, always provide lower and upper bounds on the lower and upper fine curves, which would be obtained from the original fine model. This is proved once and for all for the translation scheme.

When analyzing a coarse model, we get a pair of coarse output arrival curves that already provides lower and upper bounds on the arrival patterns of the output stream at fine granularity, by applying a simple rescaling. But it is possible to derive tighter fine output curve, running the analysis at different granularity levels $g_1 ... g_m$. The resulting coarse output curves $\breve{\xi}_{g_1} ... \breve{\xi}_{g_m}$ are then combined (see Figure 4.(c)) using the causality closure [2] property of RTC curves. This results in a tighter output arrival curve ($\tilde{\xi}$) at the fine granularity which is tighter than the naive combination of the curves but still equivalent to it.

To the best of our knowledge, no existing work deal with a granularity-based scheme with formal validation on the abstraction. The proposed framework complements the recent works on interfacing between RTC and TA models.

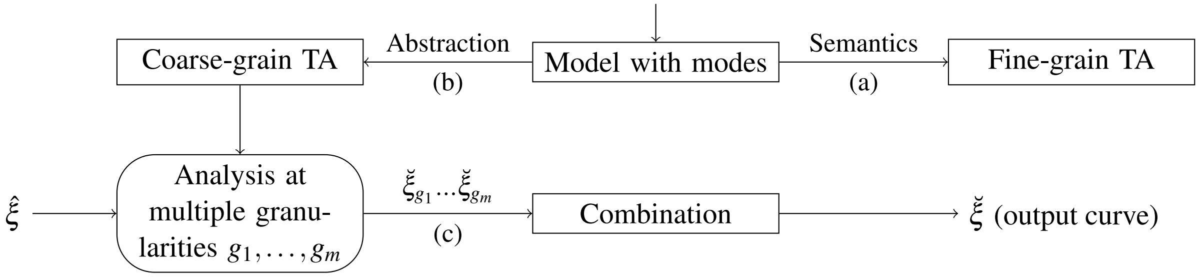

Coarse-grain TA
Abstraction
(b)
Model with modes
Semantics
(a)
Fine-grain TA
$\hat{\xi}$
Analysis at multiple granularities $g_1,\ldots,g_m$
$\breve{\xi}_{g_1}\ldots\breve{\xi}_{g_m}$
(c)
Combination
$\breve{\xi}$ (output curve)

Figure 4: The framework of granularity-based interfacing.

4 Changing the Granularity

Definitions and Notations. We denote a specific stream at the fine granularity as $\tau = (t_0)t_1t_2...$ and a specific coarse stream to be $\top = (T_0)T_1T_2....$ t_i (resp. T_i) denotes the arrival time of the i-th fine (resp. coarse) event ε_i (resp. $\mathscr{E}_i$) for $i \geq 1$. $t_0 = T_0 = 0$ denotes the origin of time. $\hat{\top}$ and $\hat{\tau}$ denote the input, while $\breve{\top}$ and $\breve{\tau}$ denotes the output stream.

As illustrated in Figure 5, we can abstract a fine stream to a coarse one at some granularity g, by regarding g consecutive fine events as a coarse one. A fine stream τ is a *refinement* of a coarse one $\top$ if T_i is equal to t_{gi} for $i \geq 0$, i.e. if it is sampling the fine stream every g events. A coarse stream $\top$ at granularity g is an *abstraction* of a fine stream if $t_{gi} = T_i$ for $i \geq 0$, the other t_k being chosen arbitrarily, with $t_{k+1} \geq t_k$. It is easy to see that a coarse stream $\top$ can be refined to a fine stream τ if and only if τ can be abstracted to $\top$.

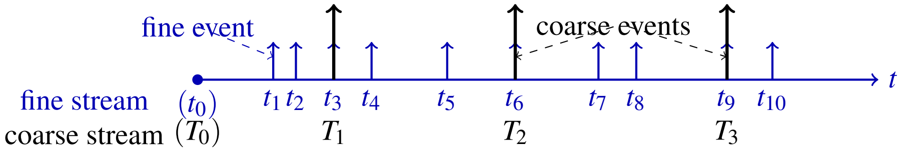


Figure 5: Illustration of a fine and coarse stream with $g = 3$.

How to obtain coarse output arrival curves? We now show how to use the interfacing between RTC and TA (Section 2) to compute output arrival curves at coarser granularity. The input arrival (resp. service) curve $\hat{\xi}$ (resp. ψ) is given for fine events. To obtain the input arrival (resp. service) curve for coarse events at granularity g, $\hat{\xi}_g$ (resp. ψ_g), we use a *sampler*. From $\hat{\xi}$ it produces $\hat{\xi}_g$ such that $\hat{\xi}_g(k) = \hat{\xi}(gk)$ for $k \geq 0$ (the definition of ψ_g is exactly the same). Indeed, it is easy to see that the arrival (resp. service) pattern for the coarse input streams are lower and upper bounded by $\hat{\xi}(gk)$ (resp. $\psi(gk)$) for all $k \geq 0$. Notice that this abstraction from fine input *streams* $\hat{\tau}$ to coarse ones $\hat{\top}$ at granularity g implies to lose information about every fine events but the ones at gi, ε_{gi}.

To compute a coarse output arrival curve $\breve{\xi}_g$, we then proceed as before: a generator is used for both the arrival curve $\hat{\xi}_g$ and the service curve ψ_g; and observers are used to compute the result using model-checking with cost optimality. Nevertheless, between them, the *component* still has to be adapted. Indeed, it was designed to proceed on fine events and needs to be abstracted to work on coarse input streams. In the sequel, we note $\mathscr{P}_0$ the fine component and $\mathscr{P}_C$ its coarse abstraction. This transformation from $\mathscr{P}_0$ to $\mathscr{P}_C$ may be hard, if possible and we define a particular class of interest, called M-TA, for which we provide an automatic transformation scheme. This is the topic of the following section. Notice that such an abstraction introduces non-determinism in the coarse component. For example, when the triggering condition of a transition depends on a number of events, the coarse grain component cannot know exactly the time at which the transition is triggered.

Validation. To validate the proposed framework, one has to guarantee that the coarse models provide *accurate abstraction* of the fine models. The proof can be made on every parts of the model but the component, since the TA are given. For the component, we exhibit a proof obligation that has to be guaranteed to validate the whole framework. This proof obligation states that the coarse component $\mathscr{P}_C$ should exhibit at least all the behaviors that the fine component $\mathscr{P}_0$ can produce. Formally, it should satisfy the following property.

Definition 1 *Let $\hat{\tau} = (\hat{t}_0)\hat{t}_1\hat{t}_2...$ ($\check{\tau} = (\check{t}_0)\check{t}_1\check{t}_2...$) be any fine event stream that is an input to (the corresponding output stream produced from) $\mathscr{P}_0$. We say that $\mathscr{P}_C$ is a* correct abstraction *of $\mathscr{P}_0$ iff(def) there always exists some coarse event stream $\hat{\top}$ ($\check{\top}$) that can be an input to (the corresponding output stream produced from) $\mathscr{P}_C$, such that $\hat{\top}$ ($\check{\top}$) can be refined to $\hat{\tau}$ ($\check{\tau}$).*

Proof Obligation 1 *Prove that $\mathscr{P}_C$ is a* correct abstraction *of $\mathscr{P}_0$.*

We will show later that the TA generated by our translation verify this proof obligation. Any other way to generate coarse TA that satisfy it could be used in the framework.

The correctness of the abstraction implies that we can derive a valid coarse output curve $\breve{\xi}_g$, by analyzing $\mathscr{P}_C$.

Lemma 1 *If Proof Obligation 1 is satisfied, it is guaranteed that the analyzed $\breve{\xi}_g^L(k)$ and $\breve{\xi}_g^U(k)$ provide lower and upper bounds on $\breve{\xi}^L(gk)$ and $\breve{\xi}^U(gk)$ respectively for $k \geq 0$.*

Sketch of proof: it follows from Proof Obligation 1 that, for any fine output stream $\check{\tau}$ from $\mathscr{P}_0$, there always exists a coarse output stream $\check{\top}$ such that $\check{\top}$ can be refined to $\check{\tau}$. It follows that the production time of the $(g \times k)$-th fine event in $\check{\tau}$ is equal to that of the k-th coarse event in $\check{\top}$. It is then easy to show that $\breve{\xi}_g^L(k) \leq \breve{\xi}^L(gk)$ and $\breve{\xi}_g^U(k) \geq \breve{\xi}^U(gk)$.

How to combine multiple coarse curves to obtain a fine one? The above paragraphs show how to obtain a coarse pair of output curves at a given granularity and prove their accurateness. We now propose to conduct *multiple runs of analysis at different granularities* $g_1, ..., g_m$, and show how to combine them to obtain a valid fine pair of output curves. The coarse output arrival curve at granularity g_i is noted $\breve{\xi}_{g_i}$ $(i = 1, ..., m)$ and we denote by $\breve{\xi}^U$ and $\breve{\xi}^L$ the optimal *fine* output arrival curves. Due to Lemma 1, we have $\breve{\xi}_{g_i}^U(k) \geq \breve{\xi}^U(kg_i)$ and $\breve{\xi}_{g_i}^L(k) \leq \breve{\xi}^L(kg_i)$ for $k \geq 0$.

Therefore, a first approximation of $\breve{\xi}^U$ (resp. $\breve{\xi}^L$) is to take the minimal (resp. maximal) obtained values for different granularities: $\breve{\xi}_{combined}^U(n) = \min_{kg_i \geq n}\{\breve{\xi}_{g_i}^U(k)\}$. But this curve is not necessarily the tightest we can find.

Suppose, for example, that we obtain output curves $\breve{\xi}_2$ and $\breve{\xi}_3$ after analyzing $\mathscr{P}_C$ with $g = 2, 3$. We can get on $\breve{\xi}(1)$ some tighter bounds than the minimum of $\breve{\xi}_2^U(1)$, $\breve{\xi}_3^U(1)$ as follows. Let $\tau = t_0t_1t_2...$ denote the trace of the fine output stream, (t_i denotes the production time of the i-th fine event ε_i), with $t_0 = 0$. Recalling that $\breve{\xi}^L(k)$ and $\breve{\xi}^U(k)$ are defined to be lower and upper bounds on the length of the time interval during which any k consecutive events are output, for any $i \geq 0$, we have $\breve{\xi}^L(2) \leq t_{i+3} - t_{i+1} \leq \breve{\xi}^U(2)$ and $\breve{\xi}^L(3) \leq t_{i+3} - t_i \leq \breve{\xi}^U(3)$. We can thus derive constraints on time interval of length 1:

$$\breve{\xi}^L(3) - \breve{\xi}^U(2) \leq t_{i+1} - t_i \leq \breve{\xi}^U(3) - \breve{\xi}^L(2)$$

Hence, a better valid bound for the time interval between any two events is given by

$$\left[\max\left\{\xi^L(1),\xi^L(3)-\xi^U(2)\right\},\min\left\{\xi^U(1),\xi^U(3)-\xi^L(2)\right\}\right]$$

Other linear constraints can be used to derive constraints in such a flavor. But, we actually use the causality closure algorithm given in [13] which refine an arbitrary pair of curves to the optimal equivalent pair of curves (provided a slight adaptation of the algorithm for finite curves to work with pseudo-inverse of the curves). This algorithm is based on the notion of deconvolution, which is basically a generalization of the above example by taking all the possible values instead of just 2 and 3 as in the example.

5 Application to Power Managed Components

The systems we target to define an automatic translation from fine to coarse models are energy-aware, or "power-managed". They have different modes of operations, in which their performance and energy consumption are defined. Most computers and embedded systems today possess energy-saving modes; for example, CPUs can have dynamic voltage and frequency scaling (DVFS) and sensors in a network can switch their radio on and off.

5.1 Models of PMC

We consider power-managed components (PMC) as systems with different modes of operation, each mode owning a pair of service curves. The system can transit from a mode to another upon various conditions. The minimal requirements to model non-trivial systems is:

(1) by receiving an explicit synchronization from another component
(2) after a given timeout (typically, systems go to a hibernation state only after spending some time in an idle state),
(3) when the buffer fill level exceeds a certain threshold (to switch to a resource-intensive one when the system is overloaded),
(4) when the buffer fill level gets below a threshold (typically, go to an energy-saving state when the buffer is empty).

Also, we need a way to force the system to stay in a given mode for a minimum amount of time (for example, to model a transition between two modes that physically takes some time). This is modeled by modifying the last two conditions to enable them only when the time spent in the current mode is greater than some constant.

Graphical Syntax to Describe PMC. From these requirements, we define the class of automata M-TA, and give a graphical syntax for it. The description is based on an enumeration of modes, each mode M_i being associated with a pair of service curves $\psi_i = (\psi_i^L, \psi_i^U)$; it is restricted to how the PMC can evolve from mode to mode. To handle the events to be computed, we suppose that the PMC is equipped with a buffer at its entry. We note q the counter for the buffer fill level: in each mode M_i it is constrained to be within some lower and upper bounds b_i^U and b_i^L. A mode M_i is also equipped with time constants L_i, U_i which represent the lower and upper time the component can stay in the mode; we use the clock $\mathbf{x}$ to measure this time: it is reset when entering a mode and checked against the bounds when going out. Figure 6 shows a descriptive model of the PMC with all the possible mode changes ((1)..(4), using the same numbering as above). We believe that our work can be easily extended to other cases of mode switch.

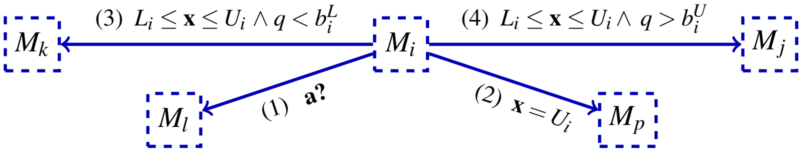
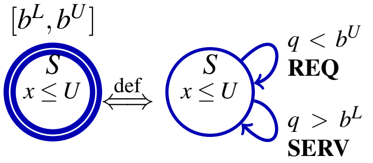

(3) $L_i \leq \mathbf{x} \leq U_i \wedge q < b_i^L$
(4) $L_i \leq \mathbf{x} \leq U_i \wedge q > b_i^U$
M_k
M_i
M_j
M_l
(1) **a?**
(2) $\mathbf{x} = U_i$
M_p

Figure 6: M-TA: 4 kinds of transitions between modes

Semantics of M-TA. Given a PMC described by a M-TA, we give its semantics in terms of the TA it represents. Using the same convention as before, a PMC receives events through the signal **req?** from the generator and emits **produce!** at its output. Internally, we translate the M-TA using two synchronized TA: the first, called *Processing Element (PE)*, controls the mode switches and the second, called *Service Model (SM)* models the computation resources for each mode. The PE and the SM are strongly synchronized: the PE emits signals $\mathbf{Syn}_{ij}$**!** whenever transiting from mode M_i to mode M_j and the SM changes its behavior accordingly. The SM has the same discrete structure as the abstract syntax. It uses one instance of the generator described in section 2 per mode M_i (i.e. per state), $Generator(\psi_i^L, \psi_i^U, \mathbf{serv})$ which emits **serv!** whenever the PMC is able to process an event. This **serv!** signal is then transmitted to the PE, which decrements its backlog q and emits a **produce!** (received by the observer).

Similarly, each time the PE receives an event to be processed via the signal **req?** (received from the generator that models the input arrival curves) this increments the backlog q. Before further detailing the implementation of the PE in terms of timed automata, we define a shortcut syntax for a state in Figure 7. The interval $[b^L, b^U]$ specified on the state S means that the system can stay in S all along the buffer fill level q is within this range. The clock invariant $\mathbf{x} \leq U$ has the usual meaning. This shortcut enables the model to be complete with respect to the incoming events **req?** and **serv?**.



REQ means: { **req?** q++; }

SERV means: `if (`q`>0)` { **serv?** q`--`; **produce!** } `else` { **serv?** }

Figure 7: Simplified notation of a state of the PE

Figure 8 shows, for the PE, the translation of the 4 kinds of transitions of the M-TA in Figure 6 into plain TA; notice that it only expands the single mode M_i. This mode is implemented with two states S_i (initial state) and S_{i1}. S_i is added to ensure that the PE doesn't leave the mode before reaching its lower timing constraint L_i. When $\mathbf{x} = L_i$, it is time to leave S_i: either the exit condition on q ($q > b_i^U$ or $q < b_i^L$) is already satisfied and the TA immediately switches to the corresponding mode (M_j or M_k) or it transits to the state S_{i1}. It can stay in S_{i1} while $q \in [b_i^L, b_i^U]$ and $\mathbf{x} \leq U_i$. Whenever q exceeds b_i^U or falls below b_i^L, it switches to a new mode (M_j or M_k). When the timeout U_i is reached, the transition to M_p is forced by the invariant. In both S_i and S_{i1}, whenever the automaton receives a synchronization signal **a?**, it must immediately transit to M_l.

5.2 PMC Models at Coarser Granularity

We now give the translation scheme from M-TAs to TA at granularity g for $g > 1$. This translation gives an abstraction of the original M-TA, which consumes and produces coarse events. Like in the previous section, we translate one PMC into two TA: the processing element (PE) and the service model (SM).

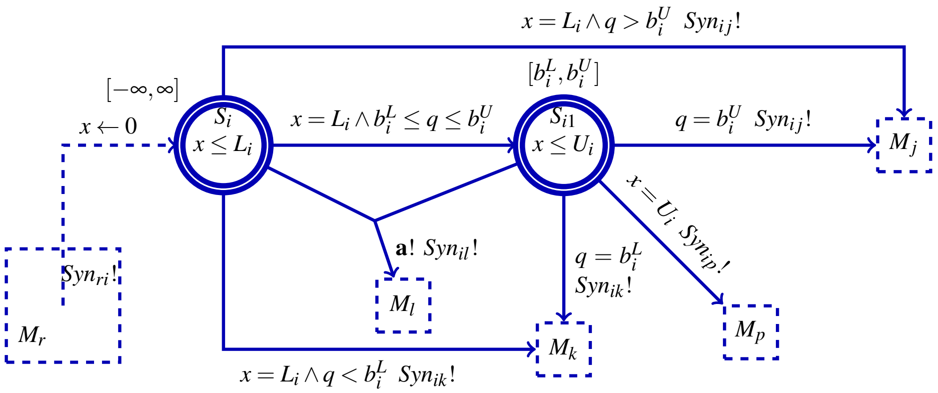


Figure 8: Fine TA model of the PE

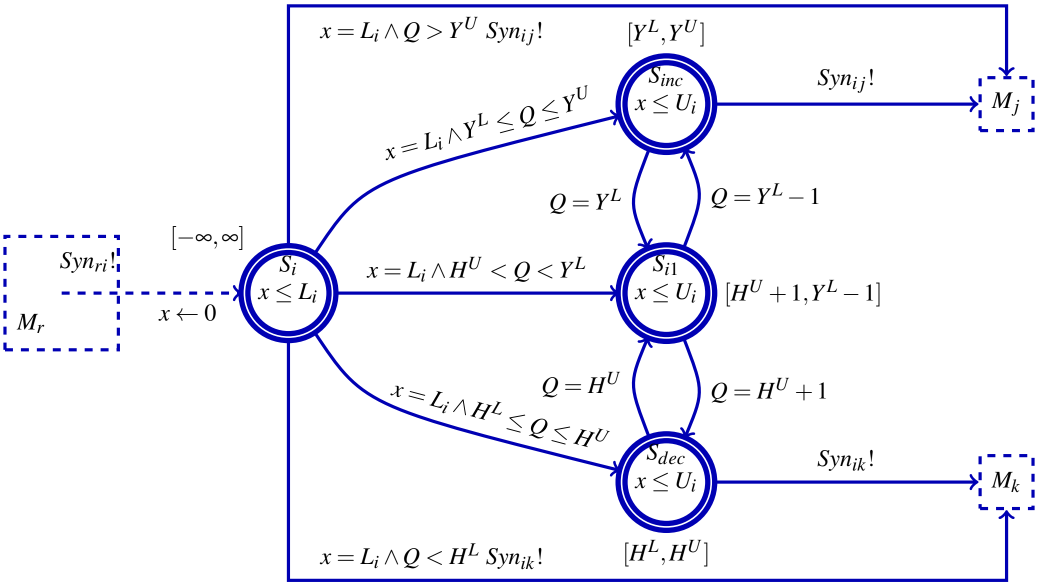
Coarse Service Model (SM). Changing the granularity for the SM is done by sampling the service curves like we did in section 4. It thus still uses one $Generator(\hat{\xi}_g^L, \hat{\xi}_g^U, \mathbf{serv})$ of Figure 2 per mode M_i but each mode is refined into two states. When a mode switch occurs, we do not know how many fine events have been emitted since the last coarse **serv!** in previous mode (but it has to be within the $[0, g)$). Arriving in the new mode M_i, the SM has to wait for k more fine events before emitting the next coarse **serv!**, k being in $(0, g]$. As shown in Figure 10, we add a state S_{trans} to capture this fact. The time the SM stays in this state is non-deterministically chosen within $[\psi_i^L(1), \psi_i^U(g)]$; it is measured by the clock $\mathbf{x}$.

Coarse Processing Element (PE). The coarse translation of the PE is given in Figure 9 using the notations introduced in Figure 7. The overall idea is the same as the fine model (Figure 8), but the buffer fill level, here noted Q, now counts coarse events instead of fine events. The state S_{i1} has been split into three states S_{i1}, S_{inc} and S_{dec} whose meaning are given in the figure. This splitting is necessary because when the condition between two modes depends on a threshold b_i^U or b_i^L on the backlog q at fine granularity, if this threshold is not a multiple of g, then the actual transition of the fine PMC would occur between two coarse events. We note Y^L and Y^U (resp. H^L and H^U) the smallest and greatest numbers of coarse events between which the fine threshold b_i^U (resp. b_i^L) can be reached. The explanation and the values on those constants are given Figure 5.2. We can therefore not determine precisely when a transition due to a threshold *does* occur in the fine PE, but we have to ensure in the coarse PE, due to the proof obligation 1, that it *can* occur at the same time as it would have in the fine PE. The coarse PE leaves the state S_{i1} when one of the transitions to M_j or M_k *can* occur, and the invariants on states S_{inc} and S_{dec} say when the transition *must* occur. The management of synchronization events ($\mathbf{a?}$) and timeout ($\mathbf{x} = U_i$) is the same as in the fine model. For clarity, it is not drawn on the TA but described above.

Validation. Due to the proof obligation 1, we now have to prove that the proposed simple coarse PMC provides a correct abstraction of the fine PMC.
Proof: let $\hat{\tau} = (\hat{t}_0)\hat{t}_1\hat{t}_2...$ be any fine input stream to the fine PMC $\mathscr{P}_0$, and $\check{\tau} = (\check{t}_0)\check{t}_1...$ is the corresponding output stream. Firstly, we construct a coarse input stream $\hat{\top} = (\hat{T}_0)\hat{T}_1\hat{T}_2...$ with $\hat{T}_j = \hat{t}_{gj}$ for $j \ge 0$. It is clear that $\hat{\top}$ can be refined to $\hat{\tau}$, while $\hat{\top}$ can be an input to the coarse PMC $\mathscr{P}_C$ due to:

$$\forall j \ge k \ge 0, \hat{\xi}_g^L(j-k) = \hat{\xi}^L(g(j-k)) \le \hat{t}_{gj} - \hat{t}_{gk} \le \hat{\xi}^U(g(j-k)) = \hat{\xi}_g^U(j-k)$$

Then we show how to construct a coarse output stream $\check{\top}$ from $\mathscr{P}_C$ corresponding to the input stream $\hat{T}$ and which is an abstraction of $\check{\tau}$: given the behavior of $\check{\tau}$, we build, by induction, the behavior $\check{\top}$ where



Coarse thresholds: $Y^L = \lfloor (b_i^U + 1)/g \rfloor$, $Y^U = \lceil (b_i^U + 1)/g \rceil$, $H^L = \lfloor (b_i^L - 1)/g \rfloor$, $H^U = \lceil (b_i^L - 1)/g \rceil$

Explanation on those values: let N_r (resp. N_s) denotes the total number of coarse *events* ***req?*** *(resp.* ***serv?****) received since the system started. Let n_r (resp. n_s) denotes the corresponding total number of* fine *events. Q (resp. q) denotes the* coarse *(resp.* fine*) buffer fill level. At any time, we have the following constraints: $gN_r \le n_r < g(N_r + 1)$ and $gN_s \le n_s < g(N_s + 1)$.*
Since the fine and coarse buffer fill levels q and Q satisfy $q = n_r - n_s$ and $Q = N_r - N_s$ respectively, we deduce that $g(Q-1) < q < g(Q+1)$. This implies that $\lfloor q/g \rfloor \le Q \le \lceil q/g \rceil$. When q is replaced with $b_i^U + 1$ in the former inequation, it provides lower and upper bounds $[Y^L, Y^U]$ on the value of Q. We can similarly derived the bounds $[H^L, H^U]$.

Meaning of the states: $\mathbf{S_i}$*: same as S_i in the fine PE.* $\mathbf{S_{i1}}$*: the coarse backlog corresponds to a fine buffer fill level between b_i^L and b_i^U.* $\mathbf{S_{inc}}$ */* $\mathbf{S_{dec}}$*: in the fine PE, the buffer fill level may have reached b_i^U / b_i^L.*

*Other transitions: 4 transitions labelled (**a?**,$\mathbf{Syn}_{il}$**!**) from S_i, S_{inc}, S_{i1}, S_{dec} to the mode M_l and 3 transitions labelled ($\mathbf{x} = U_i$,$\mathbf{Syn}_{ip}$**!**) from S_{inc}, S_{i1}, S_{dec} to the mode M_p.*

Figure 9: Simple coarse PE model

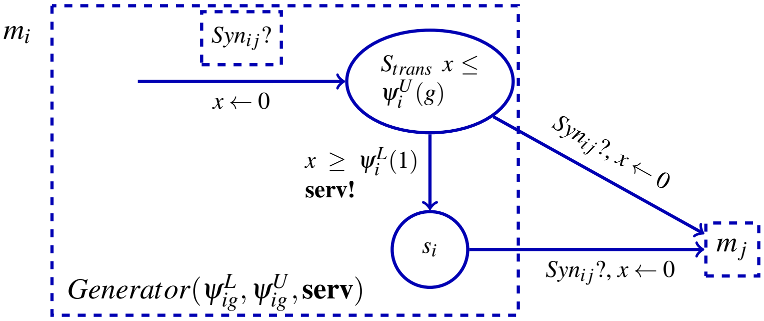

m_i
$Syn_{ij}?$
S_{trans} $x \le \psi_i^U(g)$
$x \leftarrow 0$
$x \ge \psi_i^L(1)$
serv!
$Syn_{ij}?, x \leftarrow 0$
s_i
m_j
$Syn_{ij}?, x \leftarrow 0$
$Generator(\psi_{ig}^L, \psi_{ig}^U, \mathbf{serv})$

Figure 10: Simple coarse service model

mode switches occur at the same time in the fine and the coarse PMCs.

Firstly, when the system starts, both PMCs can enter the starting mode at the same time. Suppose now that the coarse output stream $\breve{\top}$ has been constructed until the $(A-1)$-th event, for some $A > 0$, to be $(\breve{T}_0)\breve{T}_1...\breve{T}_{A-1}$ with $\breve{T}_j = \breve{t}_{gj}$ for all $j \in [0, A-1]$; and that both $\mathscr{P}_0$ and $\mathscr{P}_C$ entered the mode M_i at the same instant.

Let the fine PMC $\mathscr{P}_0$ leaves M_i when the $(gB+n)$-th fine event **produce!** is processed, with $n < g$. Two cases can then happen:

Case 1: $B = A-1$. From the service model shown in Figure 10, the time to generate the next coarse event **serv?** is lower and upper bounded by $\psi_i^L(1)$ and $\psi_i^U(g)$ respectively. It covers all the possibility of what can happen in the fine service model. Hence, it is possible that the coarse PMC leaves M_i at the same time as the fine one.

Case 2: $B > A-1$. In mode M_i, we continue to construct $\breve{\top}$ to be $...\breve{T}_{A-1}\breve{T}_A...\breve{T}_B$, with $\breve{T}_j = \breve{t}_{gj}$ for $j \in [A, B]$. Similarly to the *Case 1* above, we can show that $\breve{T}_A$ can be an output from $\mathscr{P}_C$. We can also show that $\breve{T}_{A+1}...\breve{T}_B$ can be an output from $\mathscr{P}_c$, due to $\psi_i^L(g(j-A)) \leq \breve{t}_{gj} - \breve{t}_{gA} \leq \psi_i^U(g(j-A))$ and $\psi_{gi}^L(j-A) = \psi_i^L(g(j-A))$, $\psi_i^U(g(j-A)) = \psi_{gi}^U(j-A)$. The coarse PMC is able to leave M_i at the same time as the fine PMC. Indeed, as shown in Figure 9, $\mathscr{P}_C$ may transit out from S_{inc} (or S_{dec}) anytime before Q increases to Y^U+1 (or falls below H^L-1). Hence, it is possible that $\mathscr{P}_C$ transits out at the same time as $\mathscr{P}_0$ for the case of those transitions. In other cases of transiting out, the time to transit out is same for $\mathscr{P}_C$ and $\mathscr{P}_0$, which is equal to L_i, U_i or the time receiving the synchronization signal **a?**.

It is clear that the constructed coarse output stream $\breve{\top}$ is an abstraction of the fine one $\breve{\tau}$. Therefore, the proof obligation 1 is validated.

Optimization of the coarse PMC model. This unoptimized model introduces non-determinism when mode changes occur. For example, if one knows that a mode change occurs when $q = 12$, with a granularity of $g = 5$, one can not capture in the coarse model the exact instant where q becomes equal to 12, but only the arrival of the second and third coarse events (T_2 and T_3), which correspond to $q = 10$ and $q = 15$. By reusing the information we have on the service and arrival curves at the fine granularity, we can get a better estimation than $[T_2, T_3]$ for the time at which the mode change occur. In the example, one can get a lower bound x for the time needed to get 2 more events in the buffer: $\hat{\xi}^U$ says how fast the events can arrive, and ψ^L says how fast the PMC processes them. We can compute the minimal time for which the difference between the number of events received and the number of events processed is 2. Similarly, we can get an interval $[i_s, i_e]$ on the time between the mode change and the next coarse **serv?** event. In the TA model, this is implemented by forcing the automaton to remain in the old mode for x time units after receiving the second event, and to enable the transition for the first **serv?** only when the time spent in the mode is in $[i_s, i_e]$.

We implemented two variants of this optimization in our prototype. The first introduces a counter ω and the corresponding coarse PMC is called $\mathscr{P}_C$-ω, and the second does all the complex computations on best and worst cases using this counter, and the corresponding coarse PMC is called $\mathscr{P}_C$-opt. These optimizations considerably increase the precision of the analysis. They are detailed in [12], but omitted here by lack of space.

6 Experimental Validation

Tools. To conduct experimentations on our framework, we applied the TA modeling and verification tool UPPAAL CORA [4], to model TA and to analyze the output arrival curves by model checking. The

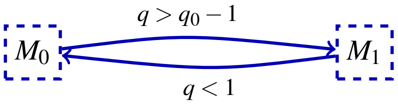


Figure 11: M-TA model of the example PMC

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\hat{\xi}^L(i)$	1	2	3	4	5	8	9	11	14	15	16	17	19	22	24	25	28	29
$\hat{\xi}^U(i)$	3	5	7	9	11	12	13	15	17	19	20	23	24	25	26	27	28	29

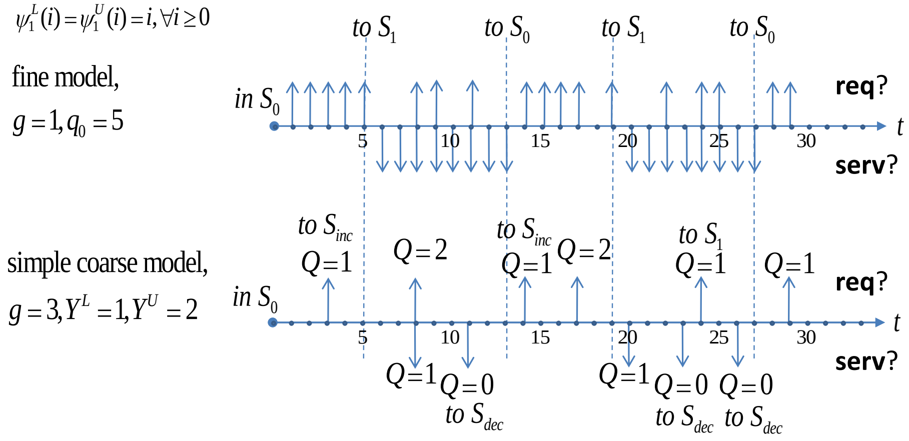


Figure 12: An illustrated flow of the example PMC

generators and observers are written once for all, but for the PMC, the M-TA translation into TA has still to be written by hand. Running the analysis at multiple granularities and the algorithm to combine the obtained curves into the tightest one is fully automated.

An Example of PMC. We illustrate the approach with the fine and coarse models of a simple PMC illustrating a common behavior. It runs at two modes: "sleep" and "run". It only processes events and consumes energy in the "run" mode and switches to the power-saving "sleep" mode when its buffer is empty. To avoid switching back and forth between "run" and "sleep", the PMC waits until its buffer fill level q reaches a threshold q_0 before transiting from "sleep" to "run". Initially the input buffer is empty (i.e. $q = 0$). Figure 11 shows the model of this example PE in terms of M-TA.

Result of the Analysis for the Example. We show the experiments conducted on the previous example where the threshold q_0 is set to 5 (i.e. the PMC starts to run when the number of fine events in the buffer reaches 5). We first apply the translation from the M-TA to TA presented above to get the coarse and fine-grain model. Notice that, since the M-TA doesn't exhibit all the cases of mode-changes, the translation can indeed be optimized manually (see [12]). In Figure 12, we take an example input stream and show its execution in the fine and coarse PMC models at granularity 3. It can be observed that the coarse PE always switches from M_0 to M_1 when it stays in S_{inc} and switches from M_1 to M_0 when it stays in S_{dec}. This illustrates how the non-determinism is introduced in the coarse model.

We now comment the results of coarse output arrival curves when running the analysis at multiple granularities $g = 2,3,4$. First the input arrival curves $\hat{\xi}$ and service curves ψ are sampled by taking the points of the curves having an abscissa multiple of g (an illustration is given in appendix A.1). Then,

output arrival curves	analysis time at granularity g [sec]									
	$g=1$	$g=2$			$g=3$			$g=4$		
	$\mathscr{P}_0$	$\mathscr{P}_C$	$\mathscr{P}_C$-ω	$\mathscr{P}_C$-opt	$\mathscr{P}_C$	$\mathscr{P}_C$-ω	$\mathscr{P}_C$-opt	$\mathscr{P}_C$	$\mathscr{P}_C$-ω	$\mathscr{P}_C$-opt
$\breve{\xi}_g^L(k)$	23674.4	56.1	62.9	223.6	7.90	12.9	189.7	0.76	1.21	28.6
$\breve{\xi'}_g^U(k)$	411515.3	746.2	583.2	4217.7	68.6	120.9	1465.4	7.43	12.5	269.1
distance	/	4.17	2.46	0.63	5.50	2.88	0.63	9.08	5.08	1.25

Table 1: Time to compute fine and coarse output arrival curves ($\breve{\xi}_g^L$, $\breve{\xi'}_g^U$), and the distance between coarse and fine curves, with $distance = mean(mean(\breve{\xi}^L(gk) - \breve{\xi}_g^L(k)), mean(\breve{\xi}_g^U(k) - \breve{\xi}^U(gk)))$ where *mean* is the average of all the elements for $1 \leq k \leq \lfloor 24/g \rfloor$.

using the UPPAAL CORA tool, we analyze the minimum cost to reach the `Stop` state of the observer model, which gives the lower output curve $\breve{\xi}_g^L(k)$ and use the variant of the observer shown in Figure 3(b) to compute $\breve{\alpha}^U$, which gives $\breve{\xi}_g^U$ after pseudo-inversion.

For granularity $g = 4$, we compare the output curves ($\breve{\xi}_4^L$, $\breve{\xi}_4^U$) using the unoptimized PMC $\mathscr{P}_C$, ($\breve{\xi}_4^L$-ω, $\breve{\xi}_4^U$-ω) using the model with ω, $\mathscr{P}_C$-ω and ($\breve{\xi}_4^L$-opt, $\breve{\xi}_4^U$-opt) using the optimized PMC $\mathscr{P}_c$-opt, as shown in Appendix A.2. It can be observed that $\mathscr{P}_C$-ω helps to obtain tighter coarse output curves than $\mathscr{P}_C$, which are further improved by $\mathscr{P}_C$-opt.

When analyzing the coarse output curves at each granularity, $g = 2,3,4$, obtained from the optimized coarse PMC $\mathscr{P}_C$-opt, ($\breve{\xi}_g^L$, $\breve{\xi}_g^U$), it can be observed that $\breve{\xi}_g^L(k)$ provides a lower bound on $\breve{\xi}^L(gk)$ and $\breve{\xi}_g^U(k)$ provides an upper bound on $\breve{\xi}^U(gk)$; where ($\breve{\xi}^L$, $\breve{\xi}^U$) is the fine output curves computed from the fine PMC $\mathscr{P}_0$ (see Appendix A.2).

Table 1 summarizes the analysis at multiple granularities by showing the total time to compute the fine and coarse output curves ($\breve{\xi}_g^L(k)$, $\breve{\xi}_g^U(k)$) for $g = 1,2,3,4$ and $k \leq \lfloor 24/g \rfloor$. It also shows the *distance* measured between coarse and fine curves. As expected, the three models $\mathscr{P}_C$, $\mathscr{P}_C$-ω and $\mathscr{P}_C$-opt allow a trade-off between performance and accuracy, $\mathscr{P}_C$ being the fastest and less precise, and $\mathscr{P}_C$-opt the slowest and most precise. The granularity g allows another trade-off: the coarsest models are the least precise ones and the fastest to analyze.

Finally, we experiment that applying the causality closure [2], (quadratic algorithm), to the resulting curves give information that neither of the analysis would have given alone. For example, running the analysis with $q_0 = 21$ at granularities $g = 9$ and $g = 10$, we get $\breve{\xi}_{10}^U(2) = 111$ and $\breve{\xi}_9^U(2) = 108$, which trivially implies $\breve{\xi}^U(10) \leq 108$. Combining the curves and using the information provided by $\breve{\xi'}^L$, we get the value $\breve{\xi}^U(10) = 102$. The complete curve is in appendix A.3.

7 Conclusion

In this paper, we have proposed a novel framework of granularity-based interfacing between RTC and TA performance models, which complements the existing work and reduces the complexity of analyzing a state-based component modeled by TA. We have illustrated the approach with an example which shows how the model of a component is abstracted to work with an event stream at coarse granularity. We did experiments that show how the abstraction is validated: they confirm that the precision of the results depends on a tradeoff with the analysis time. Indeed, the timing results show that the time to analyze the coarse models reduces at least 99% of that for the fine models. Furthermore, using the results from multiple runs of analysis at different granularities, we also demonstrate how to obtain bounds on the arrival patterns of the fine output stream with a reasonable loss of precision.

In future works, We aim at further characterizing the class of M-TA: from a theoretical point of view, we may compare it to some existing class, such as event count automata [8]; and from a practical point of view, we will go on checking if the expressivity of M-TA is enough to model more complex power-managed components. We also plan to adapt the granularity changes to other state-based models, namely the ones in [1]. On the other hand, a more challenging perspective would be to work on new state-based abstraction techniques for analyzing the time *and* energy consumption of a component.

References

[1] Karine Altisen & Matthieu Moy (2009): *Connecting Real-Time Calculus to the Synchronous Programming Language Lustre*. Technical Report TR-2009-14, Verimag.

[2] Karine Altisen & Matthieu Moy (2010): *Arrival Curves for Real-Time Calculus: the Causality Problem and its Solutions*. In: *TACAS*.

[3] Rajeev Alur & David L. Dill (1994): *A theory of timed automata*. Theoretical Computer Science 126, pp. 183–235.

[4] G. Behrmann, K.G. Larsen & J.I. Rasmussen (2005): *Optimal scheduling using priced timed automata*. ACM SIGMETRICS Performance Evaluation Review 32(4), pp. 34–40.

[5] Alessandro Bogliolo, Luca Benini, Emanuele Lattanzi & Giovanni De Micheli (2004): *Specification and Analysis of Power-Managed Systems*. Proceedings of the IEEE 92(8), pp. 1308–1346.

[6] *CATS tool*. `www.timestool.com/cats/`.

[7] S. Chakraborty, S. Künzli & L. Thiele (2003): *A General Framework for Analysing System Properties in Platform-Based Embedded System Designs*. In: *DATE*.

[8] Samarjit Chakraborty, Thi Xuan Linh Phan & P.S. Thiagarajan (2005): *Event Count Automata: A State-based Model for Stream Processing Systems*. In: *RTSS*.

[9] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter & R. Ernst (2005): *System level performance analysis - the SymTA/S approach*. IEE Proc. Computers and Digital Techniques 152(2), pp. 148–166.

[10] Simon Künzli, Arne Hamann, Rolf Ernst & Lothar Thiele (2007): *Combined Approach to System Level Performance Analysis of Embedded Systems*. In: *CODES/ISSS*.

[11] Kai Lampka, Simon Perathoner & Lothar Thiele (2009): *Analytic Real-Time Analysis and Timed Automata: A Hybrid Method for Analyzing Embedded Real-Time Systems*. In: *EMSOFT*. AC.

[12] Yanhong Liu, Karine Altisen & Matthieu Moy (2009): *Granularity-based Interfacing between RTC and Timed Automata Performance Models*. Technical Report TR-2009-10, Verimag.

[13] Matthieu Moy & Karine Altisen (2009): *Arrival Curves for Real-Time Calculus: the Causality Problem and its Solutions*. Technical Report TR-2009-15, Verimag.

[14] Simon Perathoner, Ernesto Wandeler & Lothar Thiele etc. (2007): *Influence of Different System Abstractions on the Performance Analysis of Distributed Real-Time Systems*. In: *EMSOFT*.

[15] Linh T.X. Phan, Samarjit Chakraborty & P S Thiagarajan (2008): *A Multi-mode Real-Time Calculus*. In: *RTSS*.

[16] Linh T.X. Phan, Samarjit Chakraborty, P S Thiagarajan & Lothar Thiele (2007): *Composing Functional and State-based Performance Models for Analyzing Heterogeneous Real-Time Systems*. In: *RTSS*.

[17] Simon Schliecker, Steffen Stein & Rolf Ernst (2007): *Performance analysis of complex systems by integration of dataflow graphs and compositional performance analysis*. In: *DATE*.

[18] Lothar Thiele, Samarjit Chakraborty & Martin Naedele (2000): *Real-time Calculus for Scheduling Hard Real-Time Systems*. In: Int. Symp. on Circuits and Systems ISCAS, 4. pp. 101–104.

A Arrival and Service Curves

A.1 Input Curves

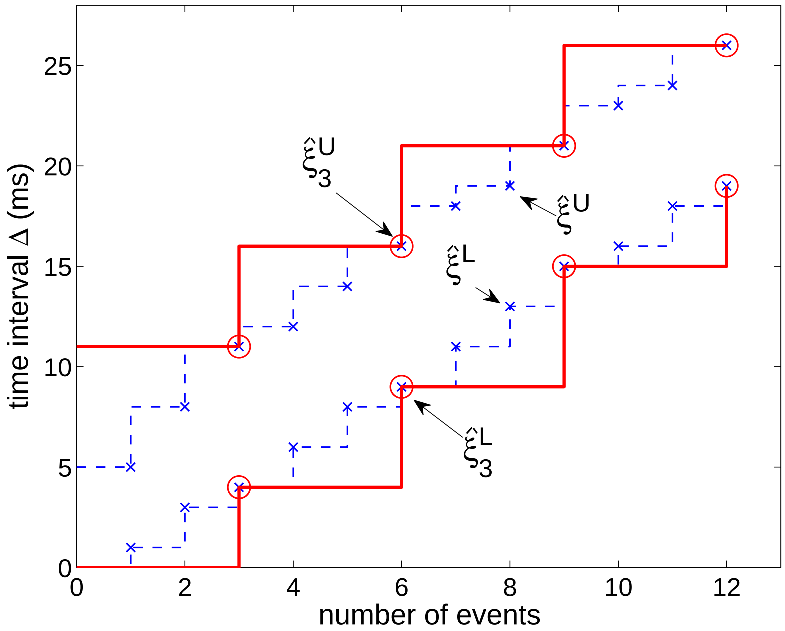


Figure 13: Example of fine and coarse input arrival curves $\hat{\xi}^{L/U}$ and $\hat{\xi}_3^{L/U}$

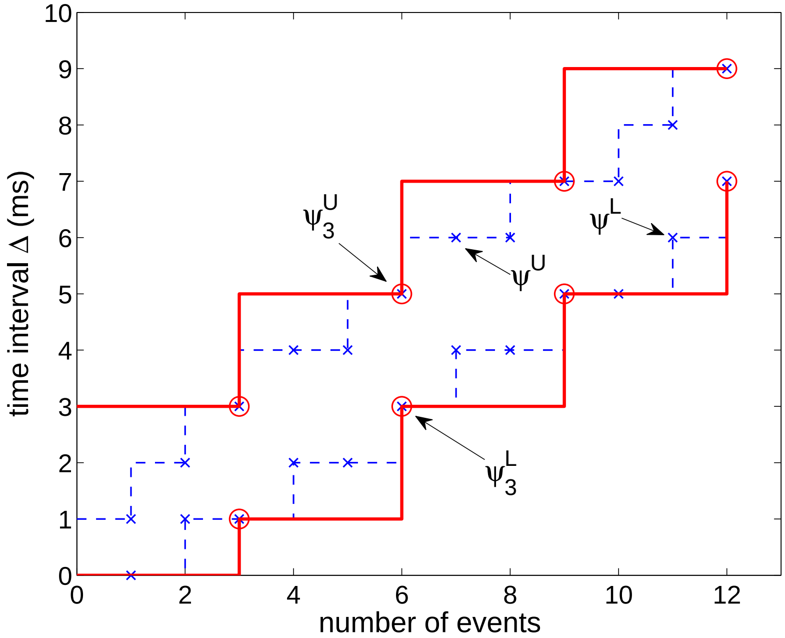


Figure 14: Example of fine and coarse service curves $\psi^{L/U}$ and $\psi_3^{L/U}$

A.2 Output Curves

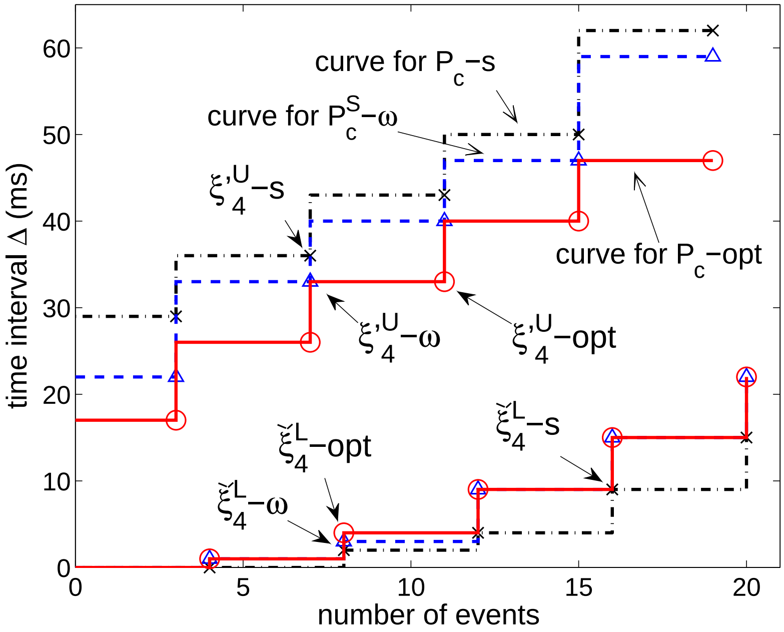


Figure 15: Comparison of output arrival curves at $g = 4$ using simple coarse PMC $\mathscr{P}_c$, coarse PMC with ω $\mathscr{P}_c$-ω and optimized PMC $\mathscr{P}_c$-opt

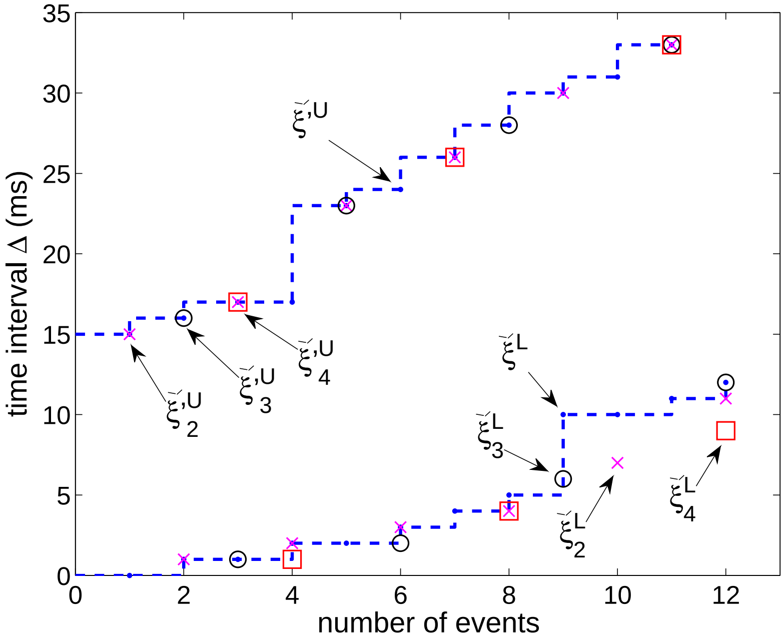


Figure 16: Comparison between fine and coarse output arrival curves computed from fine and optimized coarse PMC models respectively

A.3 Combined Output Curves

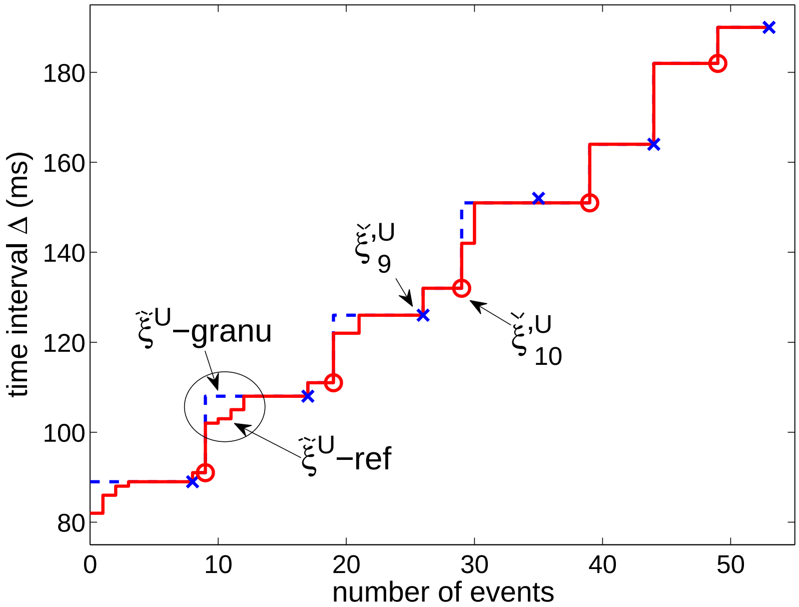


Figure 17: Computed fine upper output arrival curves $\tilde{\xi}^U$-granu (with simple scheme) and $\tilde{\xi}^U$-ref (with mathematical refinement algorithm)